Lec 16

Today: most important concept in CS — Abstraction!

Story: working on Excel — modular, but fully connected. Hard to abstract.
everything depend on everything. so little protection from other programmers
(you'll be working with idiots)

Enforcing abstraction in ML

Concept		ML
interface	—	signature
implementation	—	structure / module

```
Int.compare                    Int : INTEGER
```

Int.compare: expression; Int: module; Int : INTEGER: signature

Signature

```
Signature QUEUE =
  sig
    type 'a q          ← abstract! Client don't know what it equals to
    val empty : 'a q
    val enq : 'a q * 'a → 'a q
    val null : 'a q → bool
    val deq : 'a q → 'a * 'a q
    exception Empty
  end
```

Structure with list as queue

1. Abstraction function
   The list contains all current elems of queue in arrival order

2. Representation invariant
   None in this case

3. Code

```
Structure listQueue :> QUEUE =
  Struct
    type 'a q = 'a list
    val empty = []
    fun enq (q, x) = q @ [x]
    fun null [] = true
      | null _ = false
    exception Empty
    fun deq [] = raise Empty
      | deq x::q = (x,q)
  end
```

opaque ascription, but can make transparent by replace ":>" with ":". ↑ should only use for debugging

abstract type

need to match signature

← can write things other than those in signature

```
Structure Q = ListQueue
```

Client

```
val q2 = Q.enq (Q.enq (Q.empty, 1), 2)
    q2 : int Q.q ← Client not allowed to treat it as int list
  = - : int Q.q ← SMLNJ prints this
val ('a, b) = Q.deq q2
val ('c, _) = Q.deq q2
val ('d, _) = Q.deq b
```

* ML by default doesn't print what you're supposed to see.

Improving queue with double stack

1. Abstraction

   (front, back) — front @ rev back contains elems of queue in arrival order

3. Code

```
Structure TwoListQueue :> QUEUE =
  Struct
    type 'a q = 'a list * 'a list
    val empty = ([],[])
    fun enq((f,b),x) = (f, x::b)                    ← O(1)
    fun null ([],[]) = true
      | null   _     = false
    exception Empty
    fun deq ([],[]) = raise Empty
      | deq ([], b) = deq (rev b, [])               ← Better amortised O(1)
      | deq (x::f, b) = (x, (f,b))
  end

Structure Q = TwoListQueue
```

Dictionary

```
Signature DICT =
  Sig
    type key = string          ← concrete type
    type 'a entry = key * 'a   ← concrete type
    type 'a dict               ← abstract
    val empty : 'a dict
    val lookup : 'a dict → key → 'a option
    val insert : 'a dict → 'a entry → 'a dict
  end
```

Bad impl – list as dict [omit]

Better impl – tree

1. Abstraction function ← refers to mapping btwn abstract repr and theoretical data
   (key, value) pairs in tree correspond to entries in dict

2. Repr invar
   tree is sorted

3. Impl

```
Structure BST :> DICT =
  Struct
    type key = string
    type 'a entry = key * 'a
    datatype 'a tree = Empty | Node of 'a tree * 'a entry * 'a tree
    type 'a dict = 'a tree
    val empty = Empty
    fun insert (Empty, e) = Node (Empty, e, Empty)
      | insert (Node (L, e' as (k', _), R), e as (k, _)) =
          (case String.compare (k, k') of
             EQUAL => Node (L, e, R)
           | LESS => Node (insert (L, e), e', R)
           | GREATER => Node (L, e', insert (R, e))
          )
    :
  end
```

New field not in signature. But if transparent, it becomes broken as invisible fields in sig can get printed.

could define as datatype in struct

① Don't pattern match structure